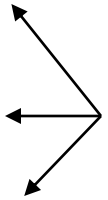
Week 4

- Repetition with for loop
- Using nested loops
- Repetition with while loop
- Repetition with do - while loop
- Using break and continue
- Program Testing
- Input Validation
- Introduction to Arrays
- Command Line Arguments

Read Pages 96 – 119
And
Pages 170-173

Repetition - repeating a set of actions a number of times.

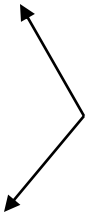
- Sum 1 + 2 + 3 + 4 + 5 + … 50
- Find n! = 1 * 2 * 3 * 4 * 5 * 6 *…n
- m^n = m * m* m …. n-times



Number of times known in advance

- Definite

- Reading and adding the numbers until user enters a special value
- Reading and adding input numbers until sum exceeds set value

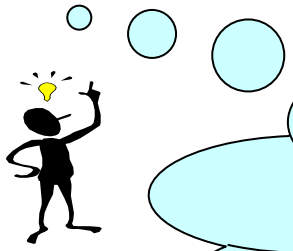


Number of times to repeat cannot be determined

- Indefinite

Finding average of 3 marks input

```
CALCULATION OF ASSIGNMENT AVERAGE
Enter mark of assignment 1:  5
Enter mark of assignment 2:  8
Enter mark of assignment 3: 10


The average mark is: 7.66
```



Design

calculate the sum of the marks

process the average

calculate the sum of marks

for all 3 assignments

prompt for mark

get mark

add to sum

```
Enter mark of assignment 1:  5
Enter mark of assignment 2:  8
Enter mark of assignment 3: 10
```

num

~~5~~ ~~8~~ 10

sum

~~0~~ ~~5~~ ~~13~~ 23

average

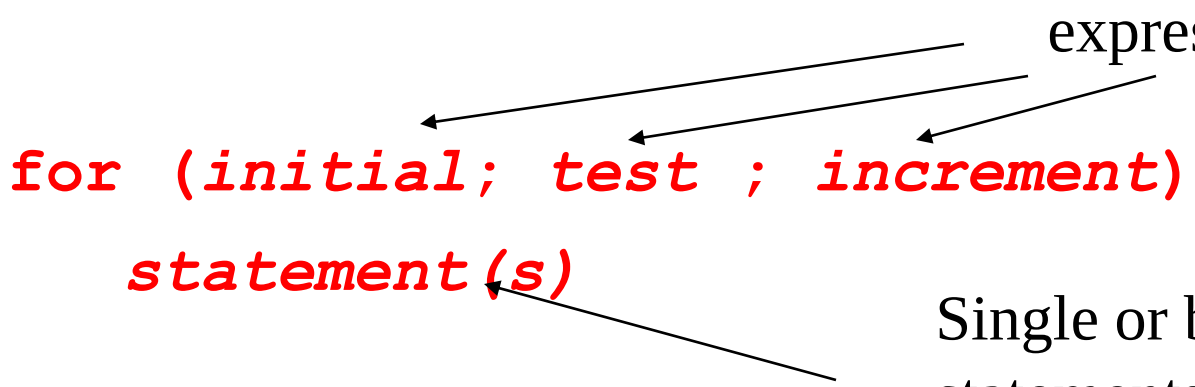
7.66

```
process the average

    calculate the average

    display the average
```

```
Enter mark of assignment 1:  5
Enter mark of assignment 2:  8
Enter mark of assignment 3: 10
The average mark is: 7.66
```

```
import java.io.*;
public class AverageMarks {
  public static void main (String[] args)
                                    throws IOException  {
    BufferedReader stdin = new BufferedReader
                     (new InputStreamReader (System.in));
    String inString;
    int i, num, sum = 0;
    for ( i = 1 ; i <= 3 ; i++)  {
      System.out.println("Enter mark of assign" + i + " : ");
      inString = stdin.readLine();
      num = Integer.parseInt (inString);
      sum += num;  // accumulate total
    }
    System.out.println("The average is: " + (double)sum / 3);
  }
}
```

General form of for loop

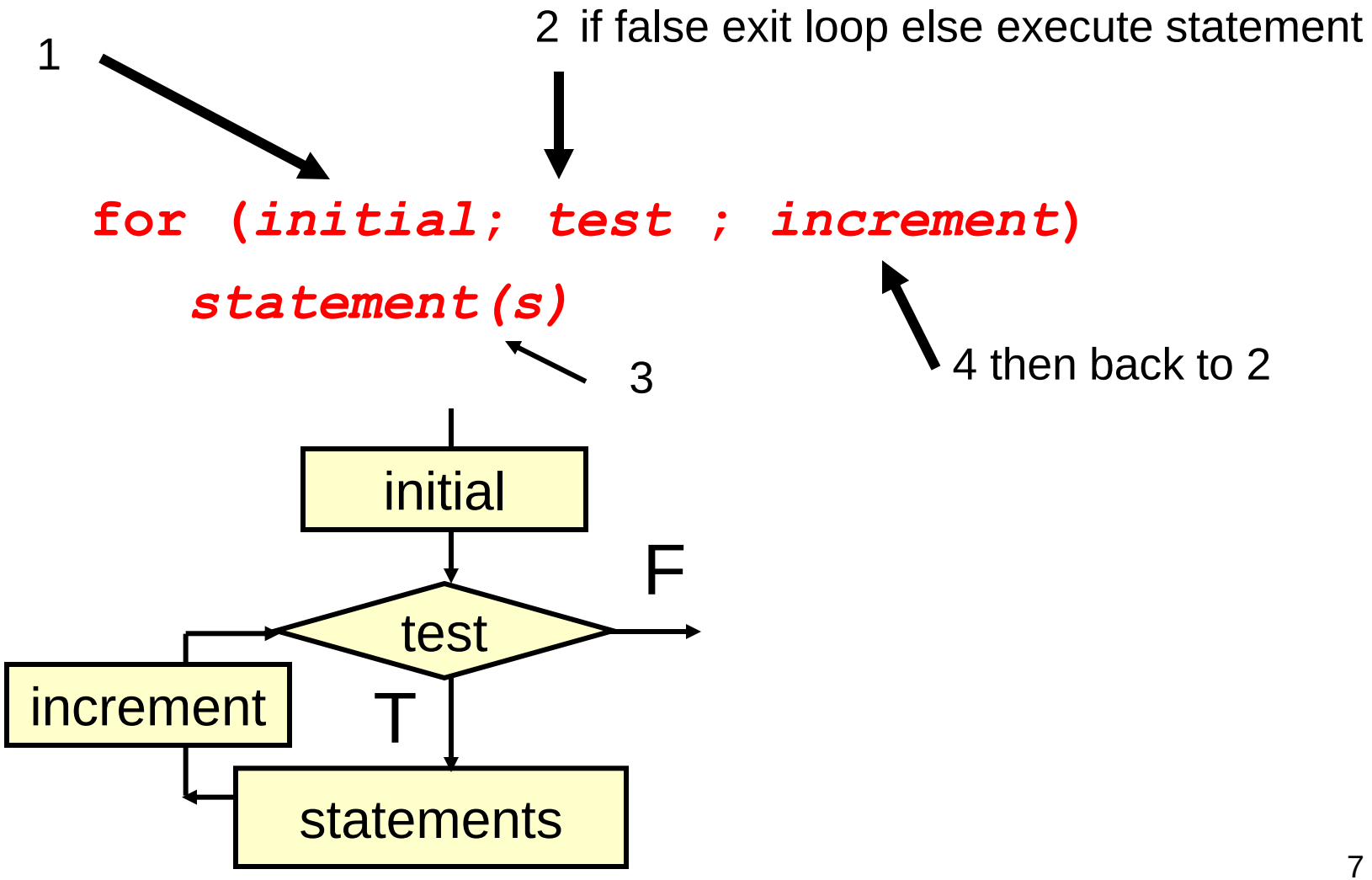



```
int sum = 0;
for (int i=0; i<3; i++){
    sum = sum + i;
    System.out.println(i);
}
System.out.println(i);
```

Convenient to declare loop variable here but lifetime is restricted to the loop

Error !

Behavior of for loop



What is the output ? (if any)

```
for (int i=3; i<3; i--)
   System.out.println("i is now "+ i);
```

```
for (int i=3; i>0; i--)
   System.out.println("i is now "+ i);
```

```
for (int i=0; i<3; i++)
   System.out.println(i*10);
```

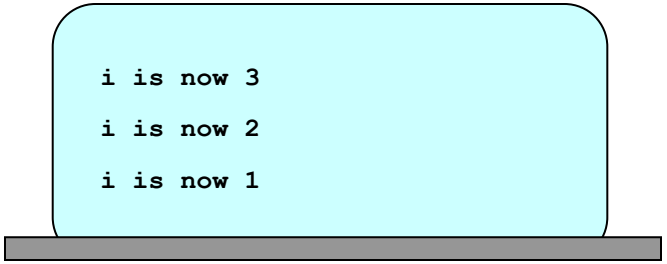
```
for (int i=0; i<3;  )
   System.out.println("i is now "+ i);
```

Output ...

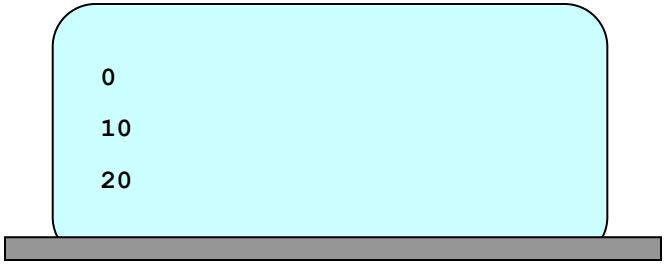
1

No output

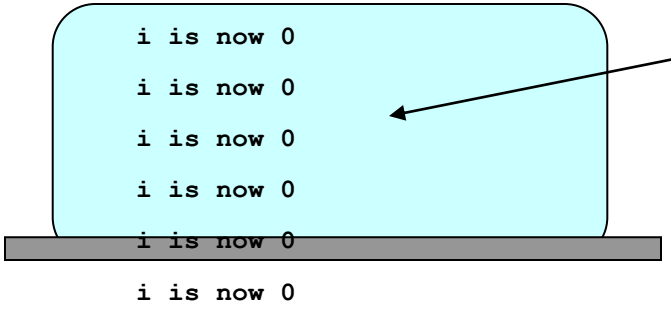
2



3



4



infinite loop

Attempting to sum 1 to 10. What's wrong ?

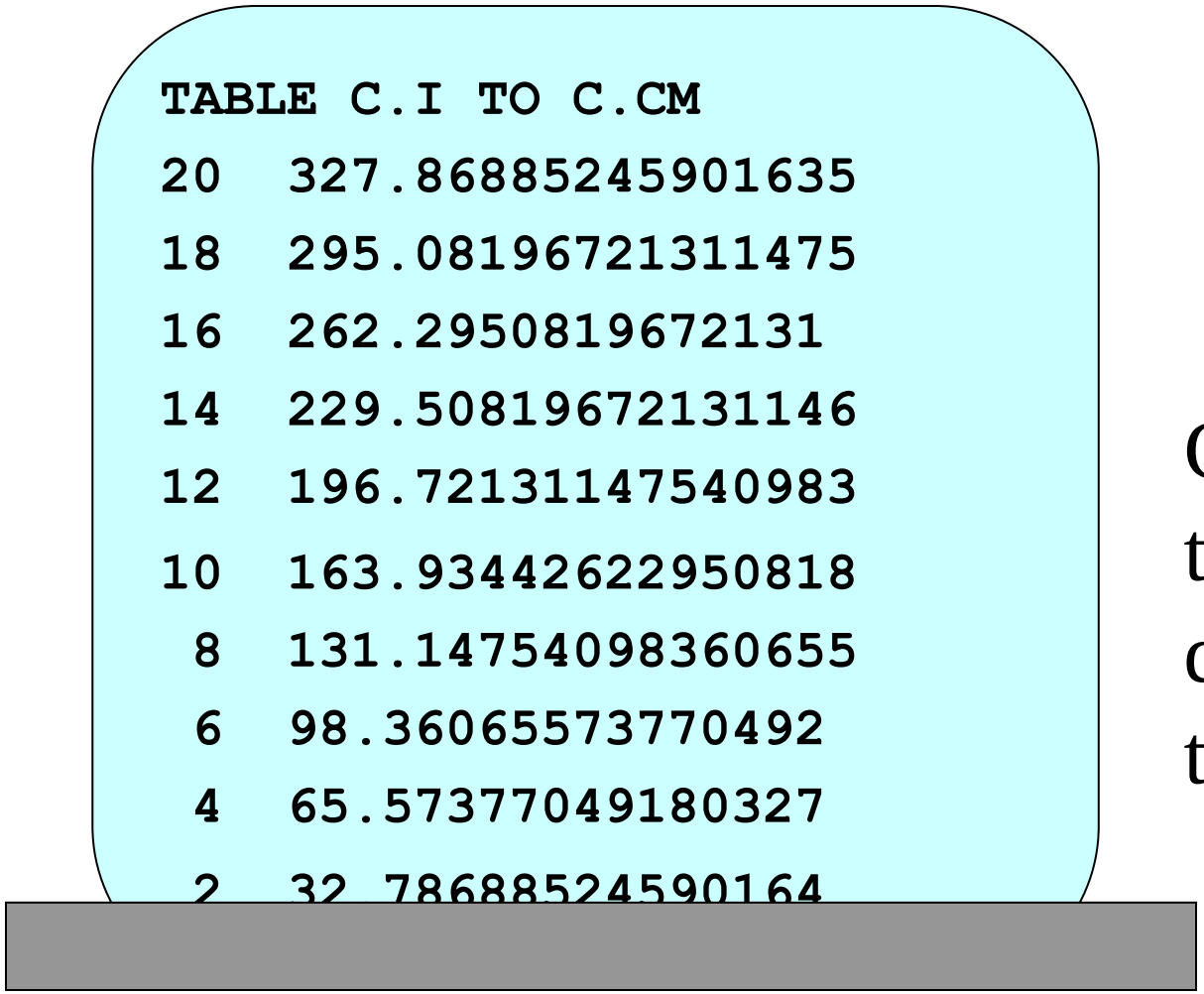
```
int sum = 0;
for (int i=1; i<=10; i++)
   System.out.println("i is now "+ i);
   sum = sum + i;
System.out.println("1 + 2..10 is " + sum);
```

```
int sum = 0;
for (int i=1; i<=10; i++); {
   System.out.println("i is now "+ i);
   sum = sum + i;
}
System.out.println("1 + 2..10 is " + sum);
```

A Conversion Table

- **from cubic inches to cubic centimeters**
- **from 20 cubic inches down to 2, in step of 2**

```
double cubicCm;
int finish, step;
// constant conversion factor C.I to C.C
final double CONV_FACTOR = 1000.0/61.0;

finish = 10; step = 2;
System.out.println(”TABLE C.I TO C.C");
for ( int i = finish; i > 0 ; i--)
{  cubicCm = (i * step) * CONV_FACTOR;
   System.out.println((i*step) + “   “ + cubicCm);
}
```



Can I restrict the number of decimal places to two ?

```
import java.text.*;
...
DecimalFormat df = newDecimalFormat("0.00");
System.out.println(df.format(cubicCm));
```

Exercise 1: Application - for loop

Complete the program to print the following series using for loops.

- 5 10 15 ……. 100
- 105 110 115 120 … 200
- 100 98 96 … 2
- 20 -20 20 -20 20 -20 20 -20 20 -20

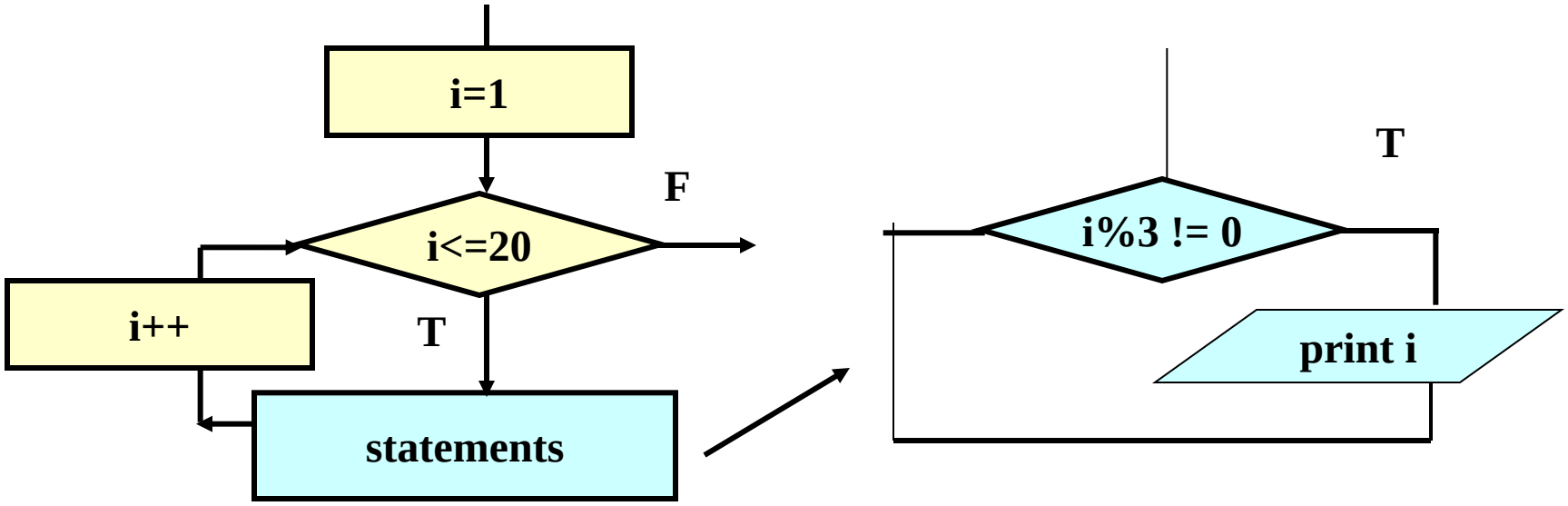
(Subsequent number formed by multiplying by -1)

Exercise 1: Application - for loop

```
public class ForLoop1
{  public static void main(String args[])
   {  // Printing 5 10 15 .... 100
      for (


      // Printing 105 110 115 .... 200
      for (


      // Printing 100 98 96 ... 2
      for (


      // Printing 20 -20 20 -20 20 -20 20 -20 20 -20
      int num = 20;
      for (int i=1; i<=10; i++)
      {


      }
   }
}
```

Nesting an if within for loop

Can I print all the numbers not divisible by 3 in the range 1 to 20 using a for loop ?

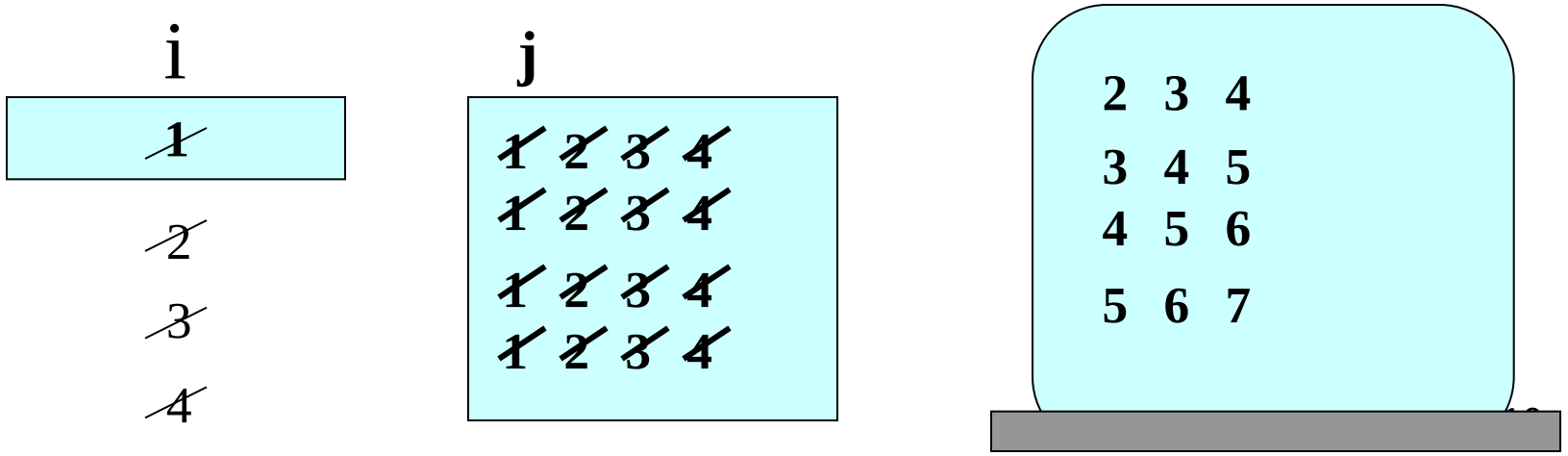



```
for ( ____; ____ ; _____)  {
   if (_____________________________)
     System.out.println();
}
```

Nested Loop

What will be the output of the program below ?

```
for (i=1; i<=4;i++) {
      for (j=1; j<=3; j++)
            System.out.print(" " + (i+j) );
      System.out.println();
}
```



Exercise: Application Nested Loop

Complete the program using nested loops to print 10*10 multiplication table:

```
1    2   3   4    5     …    10
2    4   6   8   10     …    20
…
10  20  30  40   50     …   100
```

```
public class NestedFor1
{  public static void main(String args[])
   {  for (int i=1; i<=10; i++)
      {
         for (int j=1; j<=10; j++)
         {  System.out.printf("%5d", ...);
         }
         System.out.println();
      }
   }
}
```

Outer loop
Control Variable i

inner loop

Control Variable j

Now modify program to print 12*12 multiplication table

Comparing for with while & do loops

```
public class TestLoops
{
   public static void main(String args[])
   {
      int i;
      for (i=1; i<=5; i++)
         System.out.println(i);

      i = 1;
      while ( i <= 5)
      {
         System.out.println(i);
         i++;
      }

      i = 1;
      do
      {
         System.out.println(i);
         i++;
      }
      while ( i <= 5);
   }
}
```

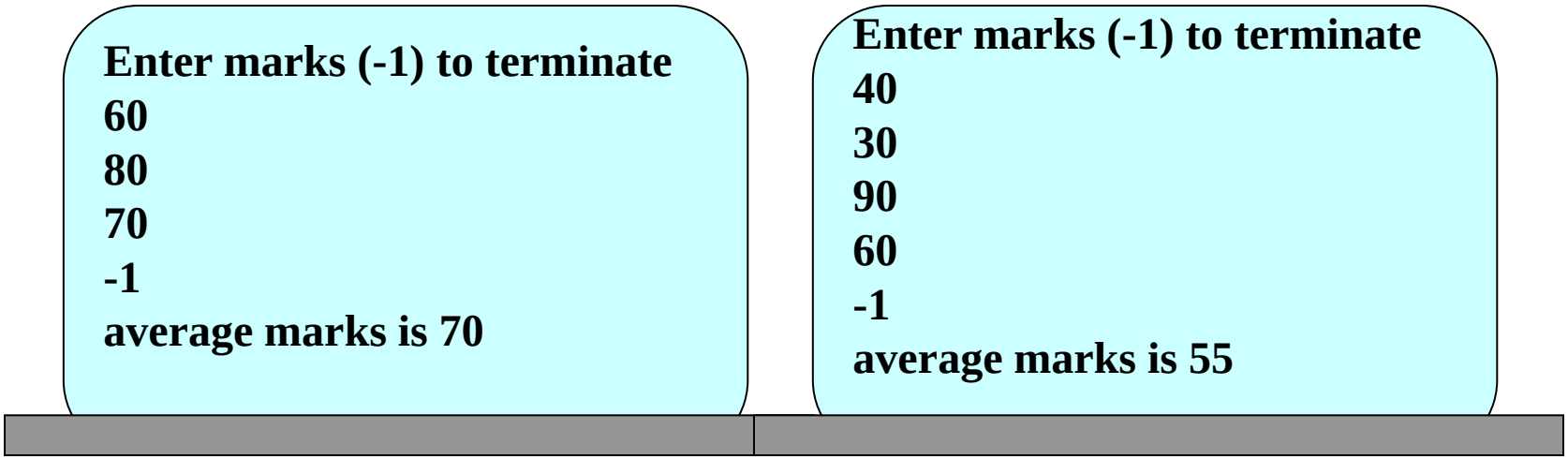
for loop

while loop

do while loop

What is the difference ?

Indefinite repetition



- Asked to write a program to find the average marks.
- Number of students in class may vary
- We must read the marks input until user enters -1
- A while loop is more appropriate

```
Initialization; // if any
while (boolean_expression)
    statement;
```

Design

```
Set sum to 0
Display Prompt “enter marks (-1) to terminate”
read mark
while (mark != -1) {
    add mark to sum
    increment number_of_students
    read next mark
}
if (number_of_students > 0) compute and print average
otherwise print error message
```

Implementation (code)

```
import java.util.*;
public class FindAverage {
   public static void main (String[] args) {
      Scanner console = new Scanner(System.in);
      int mark, sum = 0, num = 0;
      System.out.println("Enter marks (-1) to terminate");
      mark = console.nextInt();
      while ( mark != -1) {
          sum += mark;
          num++;
          mark = console.nextInt();
       }
       if (num ==0)
          System.out.println("No students in class");
       else
          System.out.println(”Aver = " +(double)sum/num);
   }

}
```

Exercise 3

The following program is attempting to read and sum all numbers until a negative number is entered. However, it gets into a infinite loop as soon as user enter a +ve number. Explain why and rectify the program.

```
import java.util.*;
public class TestWhile
{  public static void main (String[] args)
   {  int num, sum = 0;
      Scanner console = new Scanner(System.in);
      System.out.print("Enter numbers to sum.”);
      System.out.print("Enter -ve number to terminate ");
      num = console.nextInt();
      while ( num >= 0)
      {
           sum += num;
      }
      System.out.println("Sum of all +ve numbers = " + sum);
   }
}
```

Requirements Specification

Write a program to accept characters as input, and quit when a 'Q' or a 'q' is typed, printing how many characters other than 'q' or 'Q' have been typed.

```
This program counts the number of characters entered.
Input a character, 'Q' or 'q' to quit:
r
Input a character, 'Q' or 'q' to quit:
e
Input a character, 'Q' or 'q' to quit:
s
Input a character, 'Q' or 'q' to quit:
P
Input a character, 'Q' or 'q' to quit:
J
Input a character, 'Q' or 'q' to quit:
q
The number of characters entered is:    5
```

Design

```
initialisation
    show initial prompts
while input not a q or a Q
    process character
prompt for a character
        get character
        add 1 to total
display total
```

```
import java.io.*;
public class CharTest {
   public static void main (String[] args)
                                    throws IOException {
     BufferedReader charInput = new BufferedReader
                      (new InputStreamReader (System.in));
     char answer;
     int charTotal = 0;
     System.out.println("This program counts the "
                    + "number of characters entered.");
     System.out.println("Input char 'Q' ,'q' to quit:");
     answer = charInput.readLine().charAt(0);
     while (answer != 'Q' && answer != 'q')  {
         charTotal++;
        System.out.println("Input char 'Q',’q' to quit:");
        answer = charInput.readLine().charAt(0);}
     }
     System.out.println(”Num. of chars = “+ charTotal);
  }
}
```

repeated

Note the use of charAt()

Using a do - while loop instead

We could avoid the double prompting using a do-while loop and break.

```
do  {
   System.out.println("Input a character, "
                    +  "'Q' or 'q' to quit:");
   answer = charInput.readLine().charAt(0);
   if (answer == 'Q' || answer == 'q') break;
      charTotal++;
}  while (true);
```

The general form of `do-while`

```
do  {
    statements;
}  while (condition);
```

- Difference between while and do -while
- while is skipped altogether if the condition is false - hence it it termed '0 or more repetitions'
- The do loop is executed at least once, so it is called a '1 or more repetitions' loop.

Input Validation

Input validation is the process of writing extra code to ensure the user inputs only valid data.

A simple validation technique is to:

- declare input variable of an appropriate type
- prompt user for input in the valid sub-range
- check if input value is of appropriate type and within valid sub-range
- if so, continue normal processing,
- if not, re-prompt and re-check until valid

Application: Input Validation

- Complete the program to read a mark and print either pass (mark >=50) or fail (mark<50)

- Validate that the input mark entered lie in the range 0 to 100. Do this by asking the user to reenter a valid mark repeatedly until user enters a mark in the range 0 to 100.

Exercise 4: Complete Program Validating Input

```
import java.util.*;
public class Validate
{
   public static void main(String args[])
   {
      Scanner sc = new Scanner(System.in);
      int mark;
      do {
         System.out.print("Enter mark [0-100]: ");
         mark = sc.nextInt();
         if ( ……)
            System.out.print("Invalid – Reenter.”);
      } while (……);
      if ( mark >= 50 )
         System.out.print("Pass");
      else System.out.print("Fail");
   }
}
```

Exercise 5

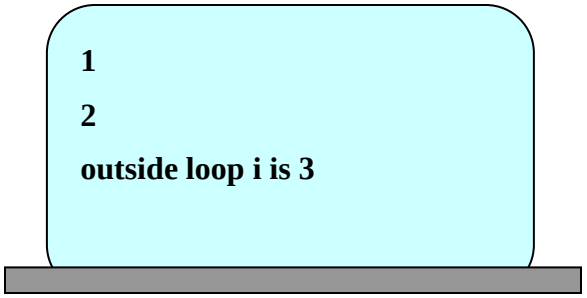
Complete the program below to read a valid gender (‘M’ or ‘F’) and then to print a welcome message in the form Welcome Ms.XXXX or Welcome Mr.xxxx. If the gender is not valid, user must be required to re-enter.

```
import java.util.*;
public class Validate
{  public static void main(String args[])
   {   Scanner console=new Scanner(System.in);
       String name;         char gender;
       System.out.print("Enter your name : ");
       name = console.nextLine();
       do {
         System.out.print("Enter gender (M/F) :");
         gender = console.nextLine().charAt(0);

       } while ( _____________________________);
       if ( gender == ‘M’)
          System.out.print(“Welcome Mr. " + name);
       else System.out.print(“Welcome Mr. " + name);
   }
}
```

The break

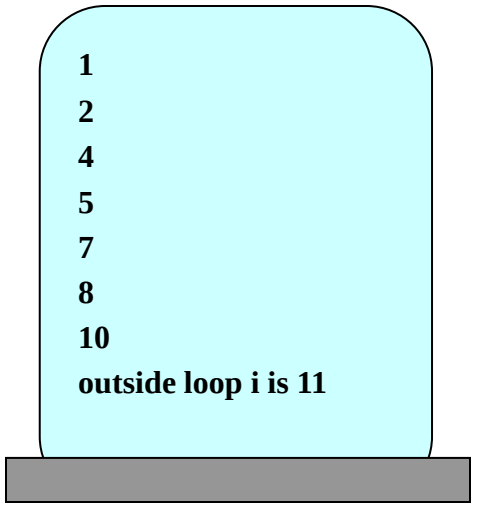
break statement allows you to break out of a loop or switch statement

```
int i;
for (i=1; i<=10; i++)
   if (i % 3 == 0)
            break;
   System.out.println(i);
}
System.out.println("Outside loop i is " + i);
```



continue statement

continue statement causes the flow of control to pass to the next iteration of the loop.

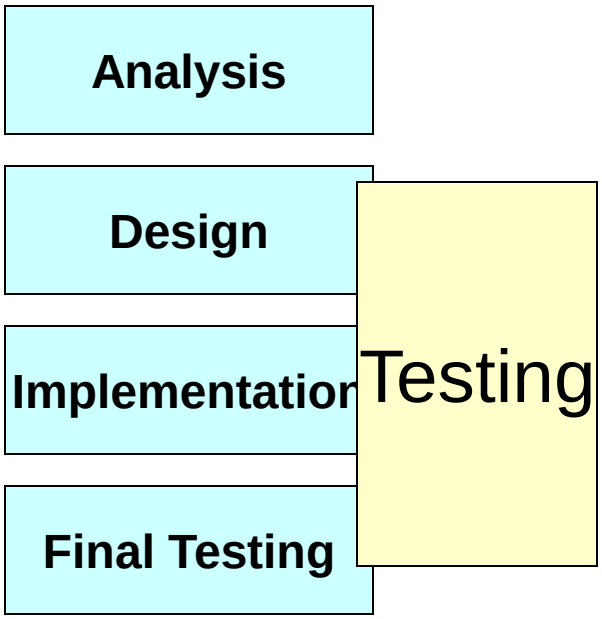



```
int i;
for (i=1; i<=10; i++) {
   if (i % 3 == 0)
            continue;
      System.out.println(i);
}
System.out.println("Outside loop i is " + i);
```

Coding Guidline:Try and avoid break and continue

Program Testing

- **Testing is intended to reveal any bugs.**
- **debugging is intended to locate and remove them.**
- **Testing is an important part of program development. it should be planned from the outset.**
- **Program testing can never prove the absence of bugs, but it can show their presence. (Dijkstra)**
- **Object oriented and modular programming facilitate testing since programs are constructed with tested building blocks (classes)**



Exhaustive testing

Consider a simple example of adding two integers (X+Y).

- We need to consider all possible input combinations.
- How long will it take to test all combinations of X + Y ?
- Java the maximum int is $2^{31}–1$ = 2,147,483,647
- Possible combinations are in billions taking years to complete the test

Black Box Testing

Black-box testing :

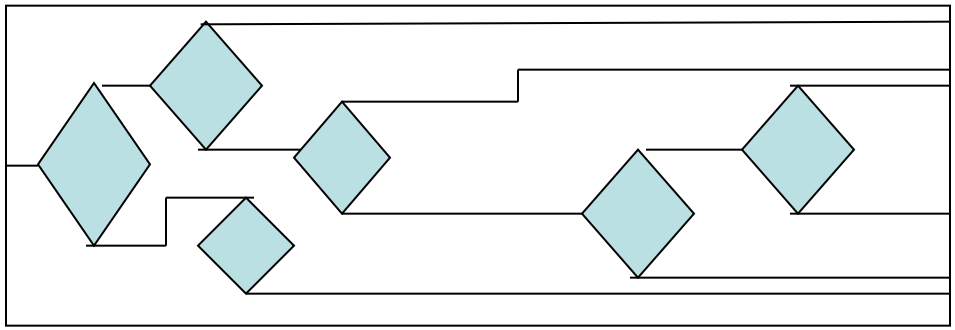
does not consider program's design structure

test plan derived from program specification

plans some specific inputs and inspect the outputs

The test values can be reused in future

White-box (Glass-box) testing:



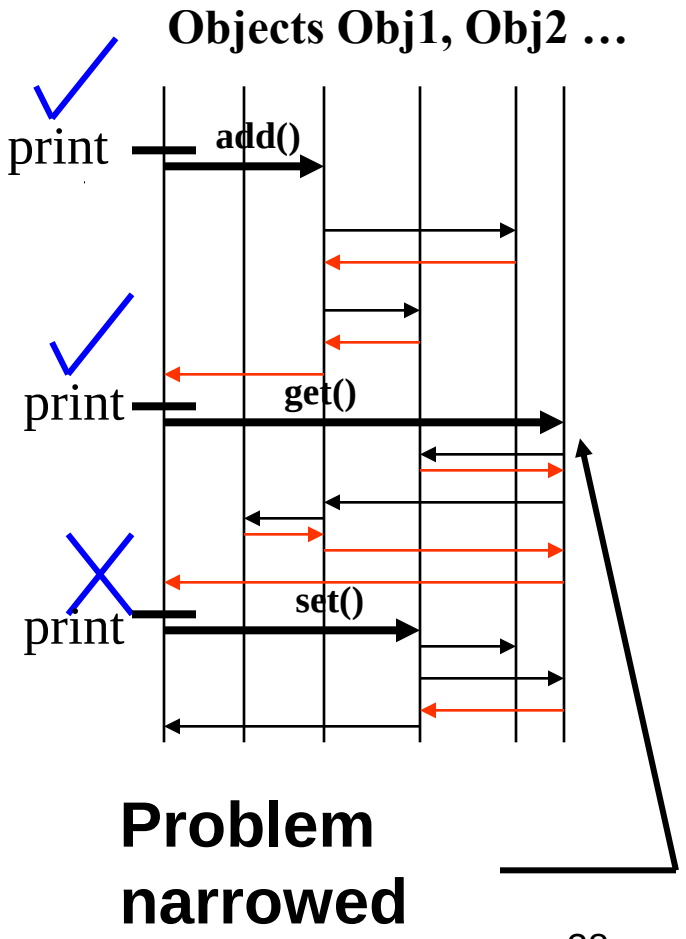
Control paths

- inputs chosen to exercise (all) control paths.
- exercises them with boundary value inputs X = a * b / (c * d)

Debugging - locating

- A basic technique to test a piece of code is to insert *diagnostic* output statement (print) after each method call.
- When the error is seen in one output but not the one prior to it, it must be between the relevant two diagnostics.
- Then we can step into the last method call trying to locate the problem. Debuggers (JBuilder, kawa) automate this process.

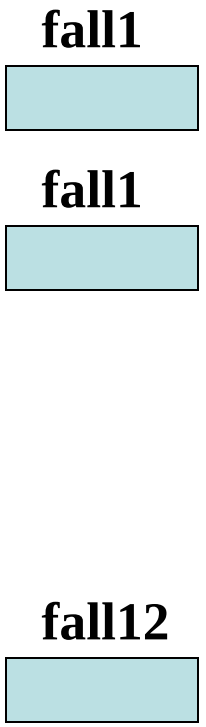
Sequence Diagram showing how the methods in various objects interact.



Why do we need arrays ?

Given the rainfall for 12 months of 2000 compute the average.

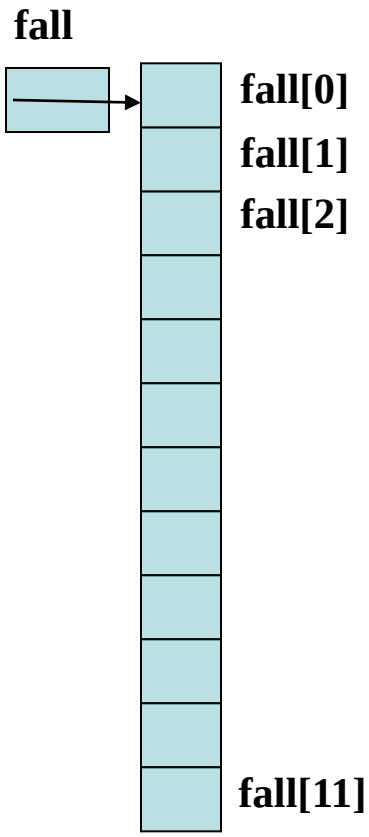
```
double fall1, fall2, fall3, …. fall12;
System.out.prinltn("R.Fall month 1 : ");
fall1 = console.nextDouble();
System.out.prinltn("R.Fall month 2 : ");
fall2 = console.nextDouble();
...
System.out.prinltn("R.Fall month 12 : ");
fall12 = console.nextDouble();
double total = fall1 + fall2 + … fall12;
double average = total/12;
```



Too many variables. Prone to errors. Not practical !

Arrays help us avoid declaring variables

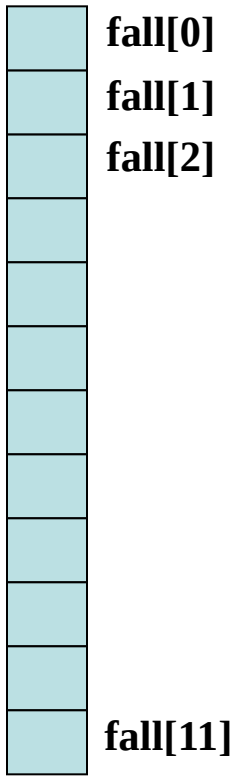
```
double[] fall;

fall = new double[12];
System.out.prinltn("R.Fall month 1 : ");
fall[0] = console.nextDouble();
System.out.prinltn("R.Fall  month 2 : ");
fall[1] = console.nextDouble();
...
System.out.prinltn("R.Fall  month 12 : ");
fall[11] = console.nextDouble();

double total = fall[0]+fall[1]... fall[11];
double average = total/12;
```



Does it make life easy ? Hardly! Many repetitive statements !

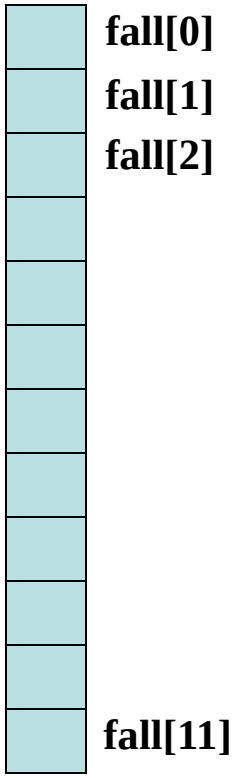
Using a loop and a control variable

```
...
double[] fall;
fall = new double[12];  // declaring array
// Reading values into array
for (int i =__ ; i < __; i++ ) {
  System.out.prinltn("R.Fall month"+i+": ");
       = console.nextDouble();
}
// Finding sum of array
double total = __;
for (int i =__ ; i < __; i++ )
  total += _________;
average = total/12;
```



Quiz

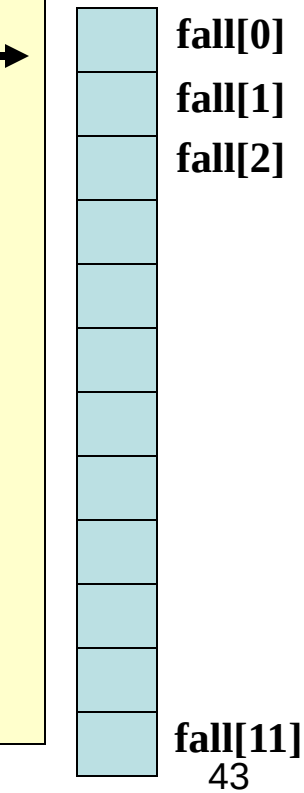
```
// sums even months Feb,Apr,June ...
for (int i = __ ; i < __; ____)
  total += fall[i];


for (int i =0; i < 7; i +=2 )
  total += fall[i];
// sums fall for months _________________
```



Exercise: Manipulating Array Elements

The program below reads and stores rainfall data from January to December. Complete the program to display them in reverse order (Dec, Nov … Jan)

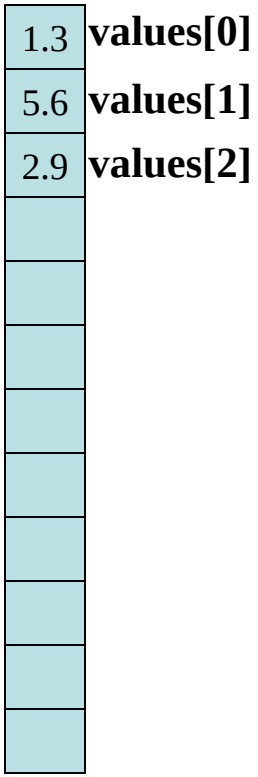
```
import java.util.*;
public class RainFall {
     public static void main(String[] args)
                              throws IOException  {
          Scanner sc = new Scanner(System.in);
          double[] fall = new double[12];
          for (int i = 0 ; i < 12 ; i++)  {
          System.out.print("Enter rainfall for month "
                                          + (i+1) + ": ");
          inString = stdin.readLine();
          fall[i] = Double.parseDouble(inString);
          }
          System.out.println("R.Fall Reverse Order ");
          for ( int i = ……….)  {
       System.out.println(……..);
          }

     }
}
```



Exercise Manipulating Array

Complete program below to print the maximum element stored in array values.

```
public class FindMax{
   public static void main (String[] args){
      double values[] = { 1.3, 5.6, 2.9, 3.7, …};
      // initially set max to first element
      max = values[0];

      // Compare max with next element.
      // If next element bigger, reset max to that

      for (.....................................)
         ..............................

      // Finally print max
      System.out.println("Sum of all numbers = " + total);
   }
}
```

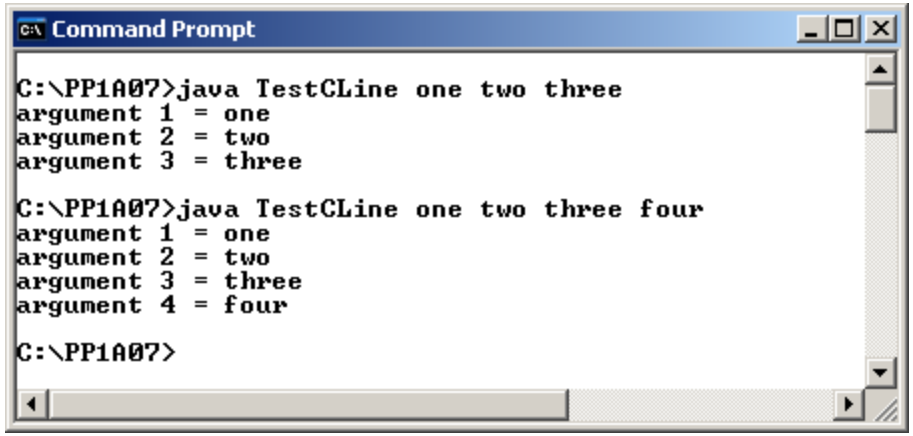


Command Line arguments

- **Recall the main method takes as arguments a reference to an array of String:** **public static void main(String[] args)**
- **Arguments can be passed from another method or command line.**
- **The following program reads all the arguments and displays them one after another. The length field is an instance of an array object.**

```
public class TestCLine{
   public static void main (String[] args)
   {  for (int i=0; i<args.length; i++)
         System.out.println("argument "+(i+1+" = "+ args[i]);
   }
}
```

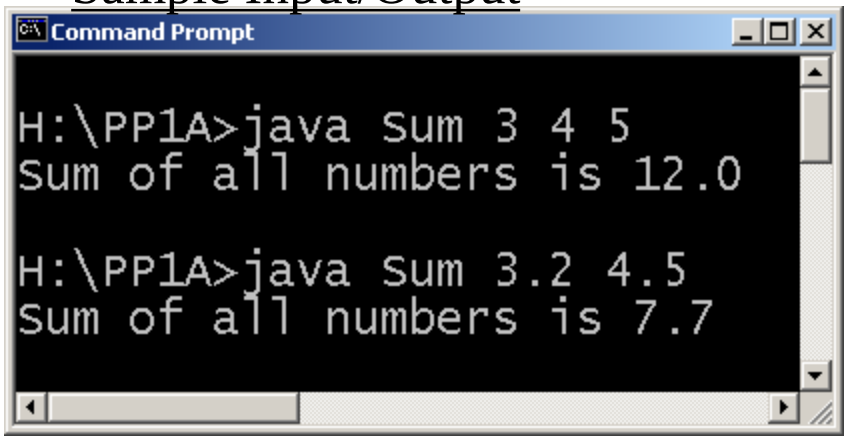
Sample Input/Output



Command Line arguments: Application

- Complete the program below for reading all numbers from command lines, adding and displaying sum.
- Hint: Double.parseDouble(String s) converts s to double equivalent.

Sample Input/Output




```
public class Sum{
   public static void main (String[] args){
       double total = 0;
       for (int i=0; i<args.length; i++)
          ...
      System.out.println("Sum of all numbers is " + total);
   }
}
```